Hw3 report

Method in details

Preprocessing

I download the bert-base-chinese-vocab.txt and use BertTokenizer in transformers as my tokenizer.

First, combine each of choice with the question into a new sentence and seperate by [SEP]. Next, feeding the article and new sentence to the tokenizer. The input is then represented like this:

`[CLS] article [SEP] question [SEP] choice [SEP]`

This returns a dictionary string to this of ints.

- **input_ids:** the indices corresponding to each token in our sentence
- **attention_mask:** point out which tokens the model should pay attention to and which ones it should not
- **token_type_ids:** indicate to the model which part of the inputs correspond to the first sentence and which part corresponds to the second sentence.

For example:

- **Sentence 1:** "How old are you?"
- **Sentence 2:** "I'm 6 years old."
- **Input:** tokenizer("How old are you?", "I'm 6 years old")
- **Output:** {
  - 'input_ids': [101, 1731, 1385, 1132, 1128, 136, 102, 146, 112, 182, 127, 1201, 1385, 102] ,
  - 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] ,
  - 'token_type_ids': [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1]

  }

In order to pad each sentence to the same length, I use parameter `padding=True` and set the max_length to 512 which the model can accept.

Sometimes, the input length is longer than the maximum sequence length of the model, so truncation is needed. At first, I use a while loop to control this condition. Compare the length of article, question and choice and delete one word of the longest each time. Keep executing until total length less than max_length(512). Then I find another solution, using parameter `truncation` can reach the same performence. I also choose the strategy ‘longest_first’ to control how both sequence in the pair are truncated.

Config


```
BertConfig {
    "architectures": [
        "BertForMaskedLM"
    ],
    "attention_probs_dropout_prob": 0.1,
    "classifier_dropout": null,
    "directionality": "bidi",
    "hidden_act": "gelu",
    "hidden_dropout_prob": 0.1,
    "hidden_size": 768,
    "id2label": {
        "0": "LABEL_0",
        "1": "LABEL_1",
        "2": "LABEL_2",
        "3": "LABEL_3"
    },
    "initializer_range": 0.02,
    "intermediate_size": 3072,
    "layer_norm_eps": 1e-12,
    "max_position_embeddings": 512,
    "model_type": "bert",
    "num_attention_heads": 12,
    "num_hidden_layers": 12,
    "pad_token_id": 0,
    "pooler_fc_size": 768,
    "pooler_num_attention_heads": 12,
    "pooler_num_fc_layers": 3,
    "pooler_size_per_head": 128,
    "pooler_type": "first_token_transform",
    "position_embedding_type": "absolute",
    "transformers_version": "4.25.1",
    "type_vocab_size": 2,
    "use_cache": true,
    "vocab_size": 21128
}
```

Model Details

In this assignment, I use the pre-trained model `bert-base-chinese` and the config is as above.

Moreover, I choose the `BertForSequenceClassification.from_pretrained('bert-base-chinese', num_labels=4)` in the transformers as my model. It will create a BERT model instance with encoder weights copied from the `bert-base-chinese` model and a randomly initialized sequence classification head on top of the encoder with an output size of 4.

- **Input**
  - **input_ids:** indices of input sequence tokens in the vocabulary
  - **attention_mask:** Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]
    - 0 for tokens that are masked
    - 1 for tokens that are not masked
  - **token_type_ids:** Segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1]
    - 0 corresponds to a article token
    - 1 corresponds to a (question+choice) token
  - **labels:** for computing the sequence classification

- **Output**
  - **loss:** classification loss

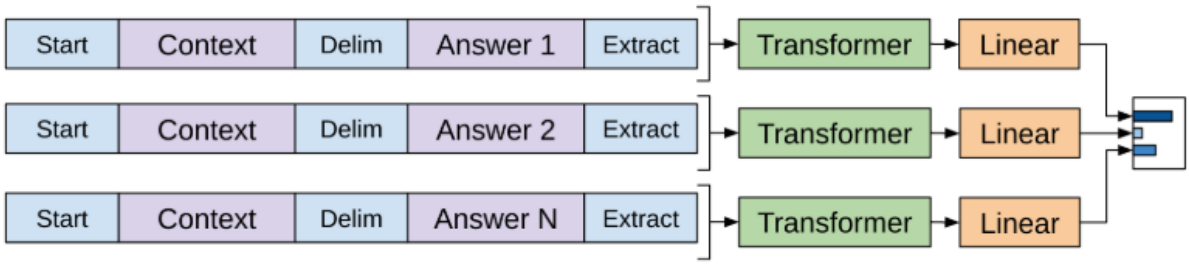
    (returned when labels is provided)
  - **logits:** classification scores before Softmax

Model Architecture

Here is the architecture of BertForSequenceClassification.

```
BertForSequenceClassification(
    BertModel(
        BertEmbeddings(
            Embedding(21128, 768, padding_idx=0)
            Embedding(512, 768)
            Embedding(2, 768)
            LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            Dropout(p=0.1, inplace=False)
        )
        BertEncoder(
            BertLayer(
                BertAttention(
                    BertSelfAttention(
                        Linear(in_features=768, out_features=768, bias=True)
                        Linear(in_features=768, out_features=768, bias=True)
                        Linear(in_features=768, out_features=768, bias=True)
                        Dropout(p=0.1, inplace=False)
                    )
                    BertSelfOutput(
                        Linear(in_features=768, out_features=768, bias=True)
                        LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                        Dropout(p=0.1, inplace=False)
                    )
                )
                BertIntermediate(
                    Linear(in_features=768, out_features=3072, bias=True)
                    GELUActivation()
                )
                BertOutput(
                    Linear(in_features=3072, out_features=768, bias=True)
                    LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                    Dropout(p=0.1, inplace=False)
                )
            )
        )
        BertPooler(
            Linear(in_features=768, out_features=768, bias=True)
            Tanh()
        )
    )
    Dropout(p=0.1, inplace=False)
    Linear(in_features=768, out_features=2, bias=True)
)
```

Process



Optimizer

And I use the `AdamW` as the optimizer which implements gradient bias correction as well as weight decay. After setting up the optimizer, themodel will do a backwards pass and update the weights.

- **params:** iterable of parameters to optimize or dictionaries defining parameter groups
- **lr:** the learning rate to use

```
no_decay = ['bias', 'LayerNorm.weight']
optimizer_parameters = [
    {
        'params': [p for n, p in model.named_parameters() if not any(nd in n for nd in
no_decay)],
        'weight_decay': 0.01
    },
    {
        'params': [p for n, p in model.named_parameters() if any(nd in n for nd in no_
decay)],
        'weight_decay': 0.0
    }
]
optimizer = AdamW(optimizer_parameters, lr=5e-5)
```

Finally, choose the the highest score in logits to be the answer.

Difficulties and Solutions

1. Sometimes, the input length is longer than the maximum sequence length of the model, so truncation is needed. Hoewever, I can't find a perfect strategy to shorten sentence length. Thus, I do some pre-processing first, for example, I delete the white-space and characters which are not Chinese.

2. I meet a problem that vocab_size in BERT config doesn't match vocab_size in BERT tokenizer, so I change the vocab_size in the config in order to fix this error.

3. At first, I use the `AutoModelForMultipleChoice` in transformers; however, accuracy is unable to increase. Therefore, I use the `BertForSequenceClassification` instead and it works.